מטלת מנחה 13 - מערכות הפעלה 20594

שאלה 2

אלגוריתמים לתזמון זרוע הדיסק מטרתם לטפל בבקשות קריאה וכתיבה מרצועות שונות בדיסק בסדר עדיפות מסוים. בשאלה זו דנים באלגוריתם LIFO, המטפל קודם כל בבקשה האחרונה.

א. היתרון במדיניות הוא מהירות הטיפול בבקשות של תהליך רץ: הבקשות מטופלות באופן מיידי בלי הצורך להמתין.

ב. החסרון במדיניות זו היא שאין חסם עליון על הזמן הלוקח על מנת למלא בקשה מסוימת. מבחינה תיאורטית, בקשה יכולה לא להיות מטופלת לעולם, כל עוד תהליכים אחרים ימשיכו לבקש בקשות - זאת משום שעל פי הגדרת האלגוריתם יטופלו קודם הבקשות החדשות ורק לאחר מכן הבקשה המקורית.

המשמעות היא שהדבר עשוי לגרום להרעבה של תהליכים.

שאלה 3

בשאלה זו נדון בשיטות שונות להקצות קבצים על פני הדיסק על מנת למלא אחר דרישות מסוימות.

א. על מנת למנוע כמה בזבוז מקום, נבחן את שלושת השיטות.

לכאורה, ב**הקצאה רציפה** לא אמור להיות מקום מבוזבז, זאת משום שכל הרצועות מוקצאות ברצף. אולם בהנחה (הסבירה) שקבצים יכולים לגדול, נצטרך להקצות אותו במקום אחר בזיכרון וכך יווצר "חור".
על פי הנתון גודל הקבצים במערכת הוא בין 4KB ו4MB. זהו טווח גדול, ולכן בסבירות גבוהה קובץ חדש לא יוכל להיות מוקצה ברציפות כולו ב"חור" הנוצר מקובץ אחר שהוזז או נמחק, והדבר יגרום לבזבוז מקום משמעותי.

גם בשיטת ה **i-node** עלול להיות מקום מבוזבז, כאשר מדובר בגודל קבצים אפשרי גדול, לעומת **רשימה מקושרת**. בקבצים בגודל קטן, אך כאלה שעדיין דורשים שימוש בשדה הindirect שב i-node, יוקצה עדיין בלוק indirect שלם שרובו המוחלט יהווה מקום מבוזבז. לעומת זאת, כאשר משתמשים ברשימות מקושרות מוקצה אך ורק מה שנדרש.

לסיכום, <u>על מנת למנוע בזבוז מקום במערכת קבצים עם קבצים בטווח גדול מאוד, נעדיף להשתמש ברשימות מקושרות</u>.

ב. על מנת למזער זמני גישה סדרתית, נבחן שוב את שלוש השיטות:

ב**רשימה מקושרת** וב**i-node** אין דרישה לסמיכות של הבלוקים של קובץ מסוים בדיסק. המשמעות היא שבגישה סדרתית, זרוע הדיסק נדרשת לזוז פעמים רבות בין בלוקים "רצופים" בקובץ.

לעומת זאת, גישה סדרתית לקבצים המוקצים ב**הקצאה רציפה** מתבצעת ללא תזוזות מיותרות של זרוע הדיסק, והדבר ממזער את זמן הגישה הסדרתית.

לסיכום, <u>על מנת למזער זמני גישה סדרתית, נעדיף להשתמש בהקצאה רציפה</u>.

שאלה 4

השאלה עוסקת בהשוואה של מודל התהליכים במערכת ההפעלה Android, ובדרך בה מערכת ההפעלה מממשת את קריאת המערכת fork.

א. בחלק המעשי אנחנו דנים בקונטיינרים. קונטיינרים עוסקים בבידוד קבוצה של תהליכים והמשאבים שלהם מתהליכים אחרים מחוץ לקונטיינר. הדבר דומה למודל האבטחה במערכת אנדרואיד, בכך שלכל אפליקציה מותאם מרחב ייחודי משלה המכיל מערכת קבצים הנגישה רק לה, והיא אינה יכולה לגשת לקבצים אחרים במערכת ההפעלה ללא הרשאה מפורשת (וגם להפך).

השוני בין המודלים הוא בגמישות שלהם: בעוד במודל הקונטיינרים יש הפרדה מוחלטת בין קונטיינרים שונים, במודל התהליכים של מערכת אנדרואיד, אפליקציות יכולות לגשת למשאבים של אפליקציות אחרות בהינתן הרשאה מתאימה מהמשתמש, ומערכת ההפעלה אף מספקת ממשק לכך.

ב. סוג הבידוד במערכת אנדרואיד הוא בידוד ברמת האפליקציה. מטרת הבידוד הוא להגן על האפליקציות אחת מפני השנייה, ובפרט מפני אפליקציות זדוניות העלולות לפגוע במשאבי המערכת. בידוד זה נעשה בעזרת מזהה UID ייחודי לכל אפליקציה.

ג. מערכת אנדרואיד משתמשת בשיטת COW בעת ביצוע fork. המשמעות היא שבעת ביצוע fork, הדפים בהם משתמש התהליך שקרא ל fork לא מועתקים ושני התהליכים, האב והבן, מתייחסים לאותם דפים. רק כאשר אחד מהם כותב אל דף מסוים, הוא מועתק במלואו ותוכנו משתנה. היתרון בשימוש בשיטה זו הוא שיפור מהירות ביצוע ה fork. הדבר חשוב במיוחד במערכת Android, שכן ביצוע fork מתרחש כאשר המשתמש פותח אפליקציה, ונדרש משוב מהיר בממשק המשתמש אשר העתקת תמונת זיכרון שלמה של תהליך עשוי להאט.

שאלה 5

השאלה עוסקת במנגנון אבטחה במערכת ההפעלה, לפיו כל תהליך שומר את רשימת היכולות שלו. כאשר הדבר מתבצע ללא תמיכת חומרה, ישנן שתי דרכים לעשות זאת, זאת ע"פ עמוד 266 במדריך הלמידה:

- שמירת רשימת היכולות בתוך מרחב זיכרון של מערכת ההפעלה. היתרון הטמון בגישה זו הוא גישה מהירה יותר, שכן אין צורך לגשת אל מטריצת ההגנה. החסרון הטמון בגישה זו הוא שכל שינוי של יכולות של תהליך חייב לעבור דרך מערכת ההפעלה, דבר הדורש זמן.
- הצפנה של רשומות האבטחה במרחב המשתמש, כך שרק מערכת ההפעלה יכולה לשנות אותן. היתרון בשיטה זו היא שהיא מתאימה למערכות מבוזרות, והחסרון שלה הוא שערכים אלה עלולים להידרס, שכן הזיכרון לא מוגן ע"י שימוש של התהליך.